Viscoelastic Fluid Simulation with OpenFOAM

Joan Marco Rimmek and Adrià Barja Romero
Universitat Politècnica de Catalunya. BarcelonaTECH.
(Dated: May 31, 2017)

Viscoelastic fluids do not behave like newtonian fluids and have to be explained in a different way than classical ones. In this report we will learn a bit about the models which explain viscoelastic fluids and we will base the project in the characterization and simulation of them acting in a particular experiment. This experiment is the oscillation of the fluid between infinite parallel planes. We are going to analyze and compare the obtained results with the theoretical behaviour.

INTRODUCTION

Complex fluids as a whole (which include viscoelastic fluids), do not behave as classical ones. They have an internal structure that makes them show non-newtonian rheological properties. They need to be studied since they are present in our every-day life. For instance, blood, sauces, creams, are examples which need to be studied in order to control their flow in industry.

OBJECTIVE

The objective of this work, was to learn to use the complex fluid simulator `OpenFOAM` and the extension `RheoTool` "an open-source toolbox based on OpenFOAM to simulate the flow of Generalized Newtonian Fluids (GNF) and viscoelastic fluids".

In order to accomplish it, we are going to simulate an experiment and study some of its non-classical properties. With the aid of these open source software we will design a grid and use the specific solver for the fluid under study. In addition, we will extract data and compare it with the theoretical results found in [1] and [2].

The fluid under study is a viscoelastic fluid. Unlike classical Newtonian fluids, which present a constant viscosity, complex fluids (and in particular, viscoelastic ones) exhibit rate dependant shear viscosity, which will cause different behaviours at different time-scales. This change in behaviour can be explained through the Deborah number, but we are not going to enter in detail on the theory. More information about complex fluids can be found in [3].

The experiment we are going to analyze is the response of the fluid to an harmonic movement between two parallel infinite plates. Since the resonant frequencies can be theoretically found [1], we are going to perform the same simulation for two resonant frequencies and one in between, to see the response for different forcings of the system.

MODEL

The theoretical model we are going to use in this simulation is known as Giesekus model. It derives from the well known Oldroyd-B model which represents a solution of a Maxwellian viscoelastic fluid. The parameters involved in this model are: λ which is the single relaxation time, the rate independent viscosity η_p, and constant viscosity η_s. Its constitutive equations for the stress tensor are:

$$\tau = \tau_s + \tau_p \tag{1}$$

$$\tau_s = \eta_s \gamma_{(1)} \tag{2}$$

$$\tau_p + \lambda \tau_{p(1)} - \alpha \frac{\lambda}{\eta_p} \{\tau_p \cdot \tau_p\} = \eta_p \gamma_{(1)} \tag{3}$$

In these equations, τ is the stress tensor and α the nonlinearity introduced in the Oldroyd-B model in order to obtain the Giesekus model. As it may seem obvious, if $\alpha = 0$ we recover the basic Oldroyd-B equations. It also has to be taken into account that this solver does not only use these constitutive equations but also solves them all along with the well-known Navier-Stokes equations for fluids.

We are not going to analyze these equations but just present them. In the simulation, these equations are efficiently implemented by the solver included in `RheoTool`, so we will not need to worry about them.

SETUP

A. General aspects

It would be great to simulate the behaviour of a fluid oscillating in a cylindrical pipe in order to compare it with the empirical results (apart from the theoretical ones). However, as we mentioned above, we will simulate the behaviour of a viscoelastic fluid subjected to a sinusoidal movement between two infinite plates as a first approximation to the cylindrical case.

The different steps we followed to perform the simulations in `OpenFOAM` along with the `RheoTool` library are explained in the next sections.

B. Geometry

First of all we need a geometry for the simulation to take place in. The geometry needs to be translated into a grid in an understandable way. In order to do so we generate a mesh using the `BlockMesh` utility with the following configuration:

We generate a square based prism of dimensions $(0.3, 0.05, 0.05)m$. The x axis is the one in which the pistons will move. For this prism to become what we want (two infinite planes) we set a symmetry condition in the z axis. We will call $a = 0.0025m$ the half-width of the distance between the infinite plates.

Furthermore, this geometry has to be discretized, as the x and y axis are the most important in our case, we set the number of divisions to be $(60, 60, 16)$. A figure of the resulting grid can be found in Annex II.

C. Boundary conditions and initial conditions

The best way we found to simulate an oscillating piston was to impose a sinusoidal movement at the inlet and the outlet for the velocity of the fluid. To do so, we will impose the following function in time at $x = 0$ (the fluid first entering the grid) and at $x = 0.3$ (first leaving the grid). In Annex II we can see the flow of the fluid at the initial time.

$$U(t) = x_0\omega_0 \cos(\omega_0 t) \tag{4}$$

Which is trivially derived from:

$$x(t) = x_0 \sin(\omega_0 t) \tag{5}$$

Where x_0 is the amplitude of the oscillation and ω_0 is its frequency. Observe that since $\cos(0) = 1$ the fluid starts in movement at the boundaries.

The initial condition for the fluid will be of zero velocity except at the boundaries.

D. Simulation parameters

The parameters used for the viscoelastic fluid are the ones which will allow us to compare with the empirical results for a specific viscoelastic fluid:

$$\lambda = 1.9s \qquad \eta_p = 64\frac{Kg}{m\cdot s}$$

$$\rho = 1050\frac{Kg}{m^3} \qquad \eta_s = 0\frac{Kg}{m\cdot s}$$

We will first simulate the linear case with $\alpha = 0$ and then $\alpha = 0.85$.

The amplitude we will use in our simulation is $x_0 = 0.001m$.

There is no specific way in the references to calculate the resonant frequencies but we can use the ones for a very similar case studied in [1]. Those can be calculated as:

$$\omega_0 = \frac{\eta_p\pi^2(1+2n)^2}{2a^2\rho}\left(1 + \frac{\eta_p\pi^2(1+2n)^2\lambda}{a^2\rho}\right)^{-\frac{1}{2}} \tag{6}$$

We will perform the simulation in two resonant cases and one in between:

$$\omega_0^1 = 11.250704Hz \qquad \omega_0^3 = 33.760313Hz$$

$$\omega_0^2 = 22Hz$$

RESULTS

E. General aspects

In this section we are going to analyze the results obtained from the simulation and compare them to theoretical external results. The main results are obtained for a linear viscoelastic fluid, so for $\alpha = 0$.

What we are studying in this section is the profile of velocity parallel to the infinite plates (namely the direction of the oscillation) along the perpendicular direction between the plates (y axis). This will allow us to easily compare the changes in the simulation. The point of study will be the center of the grid, $x = 0.15m$ and $x = 0.6m$ for the long case.

We are going to make the same measurements for different cases. On one hand we are going to study if the simulation reaches laminar flow at the center for a certain longitude, that is to say, we will simulate two different x lengths and see how this affects the results.

On the other hand we are going to study the effects of the discretization. First we are going to simulate with a thick grid and then with a thinner one.

As a last step of the whole project we are going to analyze the case with $\alpha = 0.85$.

All the numerical results are obtained after some periods of the oscillatory forcing, when the system has reached a steady state i. e. the velocity profile does not change after a integer number of periods.

F. Theoretical results

The theoretical result with which we will compare are obtained by [2]:

$$v(y,\omega) = dv\frac{kL}{kL - tanh(kL)}\left(1 - \frac{\cosh(ky)}{\cosh(kL)}\right) \tag{7}$$

$$v(y,t) = Re\{v(y,\omega)\exp(i\omega t)\} \tag{8}$$

with $dv = z_0\omega$ and $k = \sqrt{(i\omega\rho/\eta)(1+i\lambda\omega)}$.

In figure [1] we can see the plot obtained with these equations for different frequencies.

The x axis represents the position between the plane and the y axis the velocity.

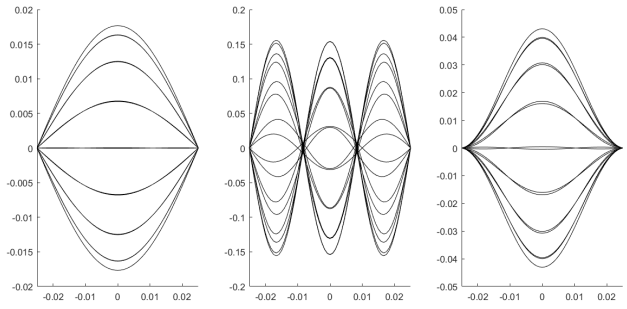

Figure 1: Theoretical U profile for different frequencies for different times in the same period. From left to right: ω_0^1, ω_0^3 and ω_0^2

G. Simulation results

The results obtained for the simulation in the same conditions has figure [1] are:

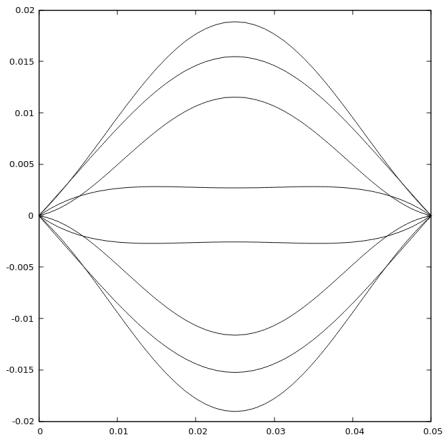

Figure 2: Simulation for ω_0^1 (1^{st} resonant frequency) for different times in the same period

We can see how ω_0^1 qualitatively behaves as the theoretical results predict whilst ω_0^2 and ω_0^3 do not match the predictions. In order to explain this we are going to analyze some variables of the simulation.

H. Longitude analysis

We are going to simulate again the second resonant frequency to observe the differences when longitude is changed. We should expect that the finiteness of the simulation will affect it and thus, elongating the grid should improve the results. However as we can see comparing figures [3] and [5] we can see a change in amplitude and also the nodes at the centre seem to displace less from $U = 0$, but the results do not behave as the theoretical ones.

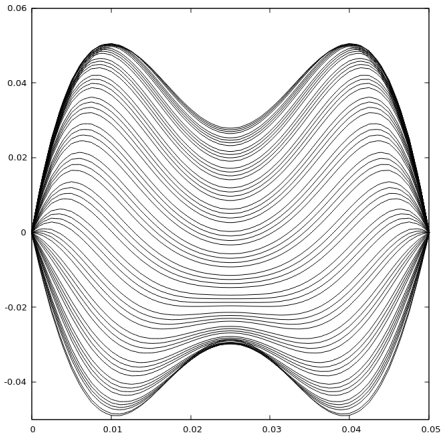

Figure 3: Simulation for ω_0^3 (2^{nd} resonant frequency) for different times in the same period

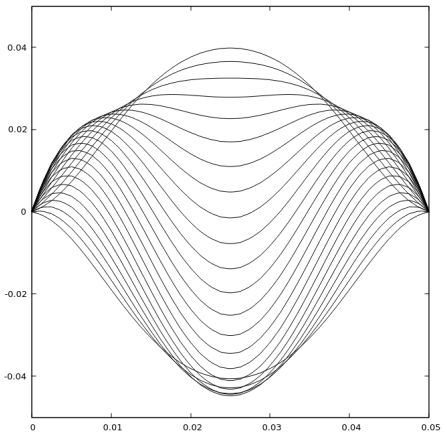

Figure 4: Simulation for ω_0^2 (non resonant frequency) for different times in the same period

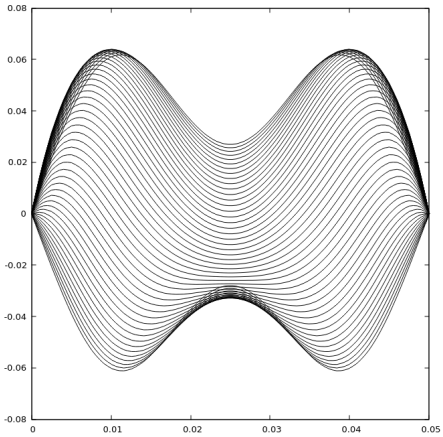

Figure 5: Longer grid simulation for ω_0^3 (2^{nd} resonant frequency) for different times in the same period

The difference in the amount of lines is there because in the moment that we performed the simulation we just

graphed a different amount of lines, it has nothing to do with the simulation in itself.

I. Fineness analysis

It is well known that in most cases increasing the fineness does not affect the results. However, in this simulation the nodes of the graph seem to displace less from $U = 0$ as we increase the number of cells in the grid. But the results still do not resemble the predictions.

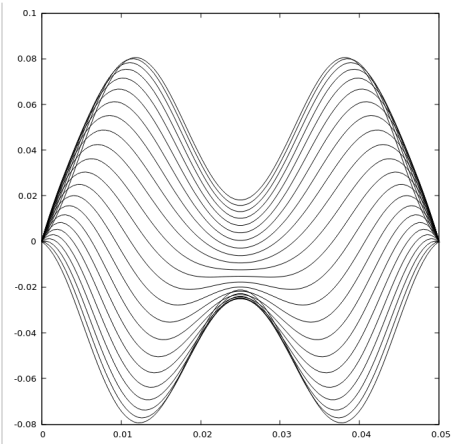

Figure 6: Thin grid simulation for ω_0^3

J. Non-linear case

Just to see how a more complex model would behave in such simulations, we simulated the non-linear case with $\alpha = 0.85$ obtaining similar results (figure [7]).

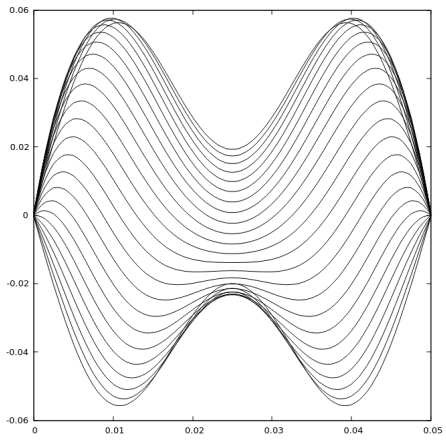

Figure 7: Simulation for ω_0^3 (2^{nd} resonant frequency) and $\alpha = 0.85$ for different times in the same period

K. Convergence

In every one of the cases aforementioned, we checked that the system reached steady state, comparing how the velocity profile evolved at the maximum velocity of oscillation in each period (in a particular direction). Obtaining a graph like [8].

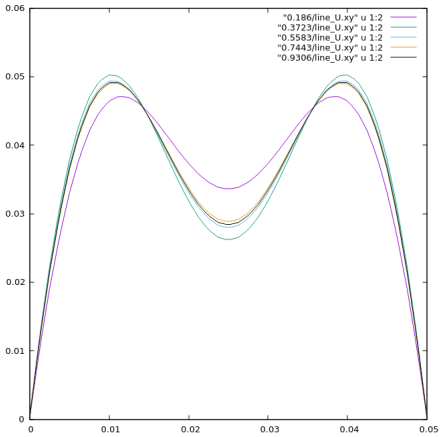

Figure 8: Convergence plot

Normally, we reached steady state after 5 or 6 periods.

CONCLUSIONS

First of all we should say that making such a simulator work was not as easy as it may seem. Installing all the dependencies, understanding how to input the model, the parameters and the grid, and then extracting the information that we can see in the plots was difficult given the limited documentation available of the software. However, we managed to do it and obtain the results shown.

The simulation results, though coherent between them, do not coincide with the theoretical results at all. We have thought of different reasons for why this could happen and reached some conclusions.

Both increasing the separation of the pistons and the thinness of the grid alters a little bit the amplitude and makes the profiles pass nearer $U = 0$ at the middle point between the parallel plates as in the theoretical results. However this does not seem to tend to the desired result. Maybe increasing the thinness of the grid a lot more could end up converging to the theoretical result but it does not look promising.

There are two main reasons that we think may explain the obtained results. On one hand, the simulator might not be doing what we wanted at all. A thorougher study and work with the software would have been great and we could have a better performance of the simulation, though it was not possible with limited time. On the other hand we may have wronged by approximating the

boundary conditions of the pistons with the function in velocities.

Finally we would like to that say we have learned a lot about the scientific method and the trial and error which characterizes it. We would also like to thank L. Ramírez-Piscina for conducting such an interesting project.

REFERENCES

[1] Laura Casanellas Vilageliu. *Oscillatory pipe flow of worm-like micellar solutions*, PhD Thesis, Physics Dept., UB. January 2013.

[2] L. Ramírez-Piscina via private communication.

[3] Alexander Morozov and Saverio E. Spagnolie. *Introduction to Complex Fluids* (book)

ANNEX I

Adrià Barja Romero and Joan Marco Rimmek

May 2017

Introduction

In this annex, the most important code files are presented. They define the properties of the fluid, the configuration of the simulation, the geomentry and mesh used and the initial conditions.

Properties of the fluid

constant/

In this folder you can find the dictionary `constitutiveProperties`.

In this file, the properties of the fluid are defined. In the example shown bellow the properties where defined by: $\eta_P = 64\frac{kg}{m \cdot s}$, $\eta_S = 0$, $\rho = 1050\frac{kg}{m^3}$ and $\lambda = 1.9s$.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                      |                                            |
| \\      /  F ield              | OpenFOAM: The Open Source CFD Toolbox      |
|  \\    /   O peration          | Version:  4.0                              |
|   \\  /    A nd                | Web:      www.OpenFOAM.org                 |
|    \\/     M anipulation       |                                            |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      constitutiveProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

parameters
{
                type            Oldroyd-BLog;

                rho             rho [1 -3 0 0 0 0 0] 1050;
                etaS            etaS [1 -1 -1 0 0 0 0] 0;
                etaP            etaP [1 -1 -1 0 0 0 0] 64;
                lambda          lambda [0 0 1 0 0 0 0] 1.9;
                uTauCoupling    true;
}

passiveScalarProperties
```

```
{
    solvePassiveScalar    off;
    D                     D [ 0 2 -1 0 0 0 0 ] 1e-9;
}
// ************************************************************************* //
```

The numbers between brackets define the units of each constant, they correspond to [kg m s K mol A cd].

Initial conditions

0/

In this folder there are four files: `p`, `tau`, `theta` and `U`.

We will show the file `U` since it is the most interesting in our case. The syntax of the others is very similar in any case.

Bellow is the initial condition for an oscillation of amplitude $0.001m$ and frequency of $33.76...rad/s$.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            codedFixedValue;
        value           uniform (0 0 0);
        redirectType    smoothU;
        code
        //Cosinusoidal
        #{
            const scalar& t = this->db().time().timeOutputValue();
            vector Uav(1, 0, 0);
            vector dirN(1, 0, 0);
            Uav = ((0.001*33.760313226149*Foam::cos(33.760313226149*t)) * dirN);
            operator == (Uav);
        #};
    }

    walls
    {
        type            fixedValue;
```

```
        value           uniform (0 0 0);
    }

    outlet
    {
        type            codedFixedValue;
        value           uniform (0 0 0);
        redirectType    smoothU;
        code
        //Cosinusoidal
        #{
            const scalar& t = this->db().time().timeOutputValue();
            vector Uav(1, 0, 0);
            vector dirN(1, 0, 0);
            Uav = ((0.001*33.760313226149*Foam::cos(33.760313226149*t)) * dirN);
            operator == (Uav);
        #};
    }

    frontAndBack
    {
        type            symmetry;
    }
}
// ************************************************************************* //
```

Mesh properties

system/

In this folder we can find the file `blockMeshDict` which dictates the geometry in which the simulation will take place.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
 (0 -0.25  -0.25)
 (3 -0.25 -0.25)
 (3 0.25 -0.25)
 (0 0.25 -0.25)
 (0 -0.25 0.25)
 (3 -0.25 0.25)
 (3 0.25 0.25)
 (0 0.25 0.25)
);

```

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (50 60 12) simpleGrading (1 1 1) //0
);

edges
(
);

boundary
(
    inlet
     {
      type patch;
      faces
      (
            (0 4 7 3)
      );
    }

    outlet
    {
        type patch;
        faces
        (
            (1 2 6 5)
        );
    }

    walls
    {
        type wall;
        faces
        (
            (2 3 7 6)
            (0 1 5 4)
        );
    }

    frontAndBack
    {
        type symmetry;
        faces
        (
            (4 5 6 7)
            (1 0 3 2)
        );
    }
);

mergePatchPairs
(
);
// ************************************************************************* //
```

The option `convertToMeters 0.1` multiplies each dimension by 0.1. Therefore, our parallel planes have a separation of $0.025m$ and the distance between the two pistons is of $0.3m$.

Simulation parameters

system/

Also, in folder system we can find the file `controlDict` which dictates the parameters of the the simulation in itself. Here you can configure the initial and final times, the time-step and personalized functions in order to retrieve data from the raw files.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     rheoFoam;

startFrom       startTime;
startTime       0;

stopAt          endTime;
endTime         0.95;

deltaT          0.0002;

writeControl    timeStep;
writeInterval   1;
purgeWrite      0;
writeFormat     ascii;
writePrecision  12;
writeCompression compressed;
timeFormat      general;
timePrecision   10;
graphFormat     raw;
runTimeModifiable no;
adjustTimeStep  off;
maxCo           0.1;
maxDeltaT       0.01;

functions
{
    #includeFunc singleGraph
}
// ************************************************************************* //
```

In this case we start the simulation at time $0s$ and end it at time $0.95s$ using a time-step of $0.0002s$. Moreover, the function `singleGraph` can be found in another file in the same folder, and is the one that retrieves the data of the velocity profiles:

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Web:      www.OpenFOAM.org
     \\/     M anipulation  |
-------------------------------------------------------------------------------
Description
    Writes graph data for specified fields along a line, specified by start
    and end points.
\*---------------------------------------------------------------------------*/

start   (0.15 -0.025 0);
end     (0.15 0.025 0);
fields  (U);

// Sampling and I/O settings
#includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"

// Override settings here, e.g.
// setConfig { type midPoint; }

// Must be last entry
#includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
// ************************************************************************* //
```

Although there are many more files in each OpenFOAM® *case*, those where the most important and the only ones we had to modify to make one simulation or another.

ANNEX II

Adrià Barja Romero and Joan Marco Rimmek

May 2017

Introduction

In this annex images and plots obtained from the simulation are present.

Grid

The grid we used in our simulations, which is $0.3m$ long and has 60 divisions in the x axis, is $0.05m$ long and has 60 divisions in the y axis and is $0.05m$ long and has 16 divisions in the z axis.

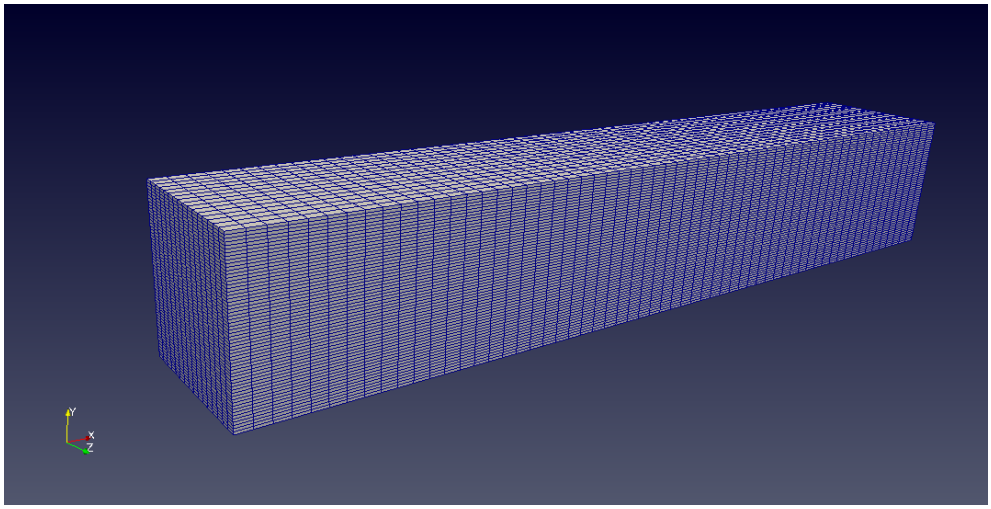

Figure 1: Grid used in the simulations

Post-processed simulations

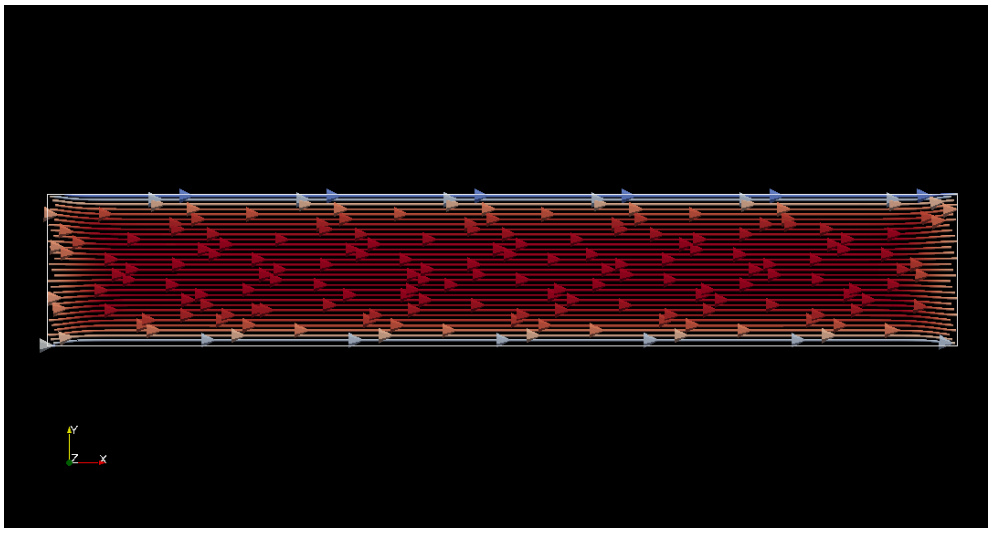

Figure 2: Image of the post-processed simulation at the first moments of the simulation

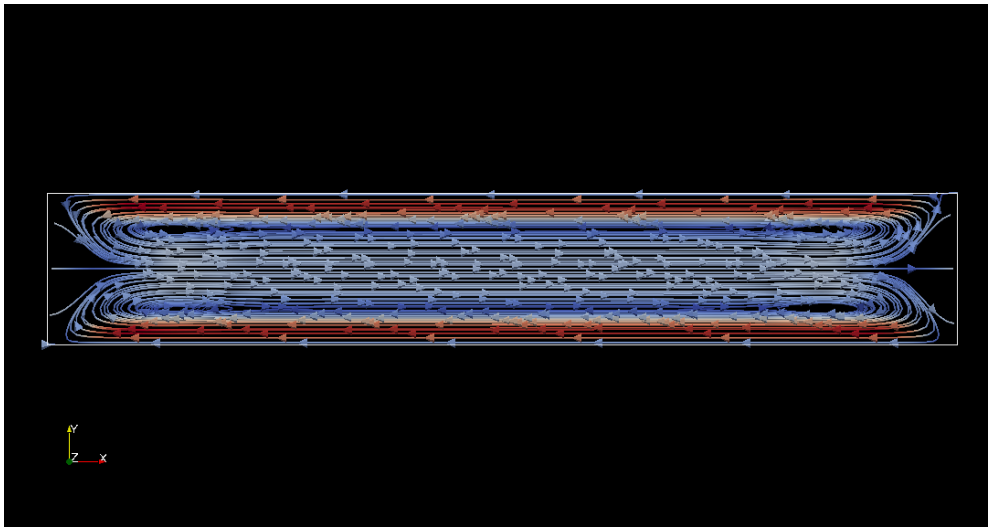

Figure 3: Image of the post-processed simulation at a change of direction in the velocity